REMARKS

Claims 1 - 3 have been rejected as obvious over a combination of US 6,199,195 to Goodwin, US 6,571,232 to Goldberg et al. and US 5,742,754 to Tse.

The invention of claim 1 differs from the combination of references in several respects. First, the invention of claim functions to automatically translate a formal language specification into a full and complete source code program which needs no further third party source code or source code from preexisting components or code libraries to complete it. The output of the process of claim 1 is a full and complete source code program which can be compiled into a full and complete object code program which can execute by itself on a computer.

In contrast, Goodwin et al. (hereafter just Goodwin) do not teach generation of a full and complete program. Goodwin teaches automatic generation of sofware objects that must be integrated into an extensible object framework to make up a complete source code program capable of being compiled into an executable program.

To bring out this difference in the claim language of claim 1, the bulk of the preamble has been moved into the main body of the claim, and the preamble language has been amended as follows:

> automatically translating a formal specification written in a formal language defining a full and complete computer program to be automatically written by a computer into a full and complete source code computer program that can be compiled into a complete, executable program which can execute by itself on a computer and needs no additional third party source code or source code from existing components or code libraries to be compiled with it to make said complete executable program and which implements the requirements of said formal specification, said formal specification defining at least classes of objects having attributes, services and relationships with other classes, ~~said specification written in a formal language,~~ comprising the following steps:

Evidence that Goodwin does not teach or suggest the automatic generation of a complete source code software program is found in the following places in the Goodwin patent. First, the title and abstract alone indicates the Goodwin invention is automatic generation

of source code objects within an extensible object framework. The automatically generated source code objects cannot be regarded as a complete and operable computer program because they must be included within an "extensible framework".

Further evidence that Goodwin does not teach automatic generation of a full system is found in the following passages from the Detailed Description Of the Preferred Embodiments section of Goodwin:

> (Col. 13, Lines 65-67 - emphasis ours)
> "Output from the code generator can be combined with other user defined codes to create an application"
>
> (Col. 14, Lines 4-9 - emphasis ours)
> "User's own code can be combined with the generated files to produce a resultant code library. The library includes user defined behaviors and support for each of the selected services for the given framework, and for interacting with the particular data model represented in the unified model".
>
> (Col. 17, Lines 10-14)
> "The fourth step is adding the code from the developer that actually implements the methods defined for the unified models, and any custom services or user defined code from the developer. (Business logic is included in this step; this step is a development time step.)."
>
> (Col. 6, Lines 37 -41)
> "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behavior and methods."

All these passages make it clear that the human code developer must supply code to work with the source code objects output from the code generator to make a complete and operable system. In contrast to the invention of claim 1, no human written code is needed. The output of the process of claim 1 is a complete and operable system.

The Examiner will note that the invention of claim 1 is not limited to the generation of any particular type of source code. In contrast, Goodwin teaches automatic generation of software code objects (Abstract, line 1, Summary of the Invention, Col. 2, line 64 to Col. 3, line 2). The Summary states:

> The present invention advantageously provides an approach for automatically generating source code, and specifically for designing and authoring source code within a complex business framework, and generating business objects with all implemented behaviors within a

composed object service framework.

And from the first line of the Background of the Invention section, we find this evidence:

> The present invention relates to automatically generated source code, and more particularly to automatically generated object-oriented source code

It is clear from these passages that the invention is not to generate just any type of source code automatically but to generate source code objects which can be integrated into an object service framework (which is not automatically generated by the Goodwin system) to make a complete, operable system. No object standing alone or in combination with all or any portion of the other objects automatically generated make a complete and operable system. The objects Goodwin generates must be integrated with other components or code libraries to provide the extensible framework, with the combination of the objects and the framework providing a complete system.

Claim 1 of the invention at bar is not limited to automatically generating source code objects. The specification that supports claim 1 and from which interpretation of claim 1 will be drawn, while mentioning object oriented source code as one embodiment of the invention, does not require that the source code which is automatically generated by the invention of claim 1 be source code objects.

User authentication and validation

Claim 1 at bar contains the following step:

> using a computer, automatically write computer code that will request user name and password, receive any responses and authenticate the user;

Goodwin does not teach the automatic generation of code of a software system to perform user authentication, i.e., to verify that the user is who he says he is. In Col. 16, lines 4-6 he does teach the authentication of "client/server programs", whereas the present invention discloses the automatic generation of source code of an application to perform the authentication of users of said application, users being defined as individuals using or interacting with the software system which is automatically generated. In other words, the method of claim 1 automatically generates a software system which requests users thereof to login and provide information from which said users can be authenticated.

Further evidence is found in Col.9, lines 17-20, where Goodwin explicitly teaches

that it is "other servers" (not users) that connect to the "code generator 330", not to any software system automatically generated by said "code generator".

Dynamically determining user privileges

Claim 1 calls for a step to write code which will automatically determine what privileges a user has in terms of what object attributes the user can see and which services the user can invoke:

> using a computer, automatically write computer code that will determine this user's privilege level and query said formal language specification and determine all object attributes this user has privilege to see and all services this user can invoke;

Goodwin does not teach the automatic generation of code of an application to dynamically determine user privileges as to what information said user is allowed to query and which services said user is allowed to execute. Col. 13, lines 26-28 teaches what input is passed to the code generator by the user ("either a model name of a unified model stored within the schema repository ... or a CORBA Interface Definition Language (IDL) file from model input") then col. 13, lines 52-56 teaches what the "code generator" can create from certain "user preferences and selections" ("the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files (CORBA, Java RMI, and/or COM) based on certain user preferences and selections"). That is, Goodwin teaches what user inputs will accept a certain "code generator" and what files said "code generator" will produce according to said inputs.

In contrast, the prosecuted application teaches the automatic generation of a software system that will dynamically and automatically determine user privileges, in terms of what information and services will be available to the user of the automatically generated software system. The claimed invention does not automatically create a system defined in terms of what user input that was supplied to the computer that automatically produced said software system. The only thing that controls what software is generated by the process of the invention are the statements in the formal specification.

Displaying a GUI to invoke services

Claim 1 at bar includes the following step regarding displaying a graphical user interface which can be used to invoke services:

> using a computer, automatically write code that displays menus options, icons or creates any other means by which a user or another process can

> invoke a service, and which receives input to invoke a particular service and responds by sending a message to the appropriate object server to invoke the service, said message including the necessary arguments for the service to execute;

In contrast, Goodwin does not teach the automatic generation of code of a software system that displays a GUI to invoke services. Evidence supporting this conclusion is found at Col. 6, lines 37-46. There Goodwin teaches what developers will be able to generate ("objects based on a framework of services") by means of the system and method disclosed therein.

> "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behaviors and methods. This is accomplished through the code generator 210, which interfaces with the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML)."

This does not teach automatic generation of objects which display a GUI by which a user can invoke services. There is no mention of invoking of services or a GUI.

There are references to "interfaces" disclosed at Col.9, Lines 20-23:

> The code generator 330 provides an Application Program Interface (API) that allows other servers to connect, identify a model to be transformed into a server, and invokes code generation for that server. The objective of the server generated by the code generator 330 is to support Next Generation Information Infrastructure (NGII) services, interfaces, graphs of objects, and data aware objects. The Application Program Interface (API) is required by the schema server 316 (described below) as an infrastructure between the various components shown.

It is apparent that this text is related to APIs or programming interfaces "required by the schema server ... as an infrastructure between the various components shown". The interfaces discussed there are not GUIs (Graphical User Interfaces) implemented by the automatically generated code.

Furthermore the only mention found in Goodwin about Graphical User Interfaces are:

> (Col.5, Lines 48-49)
> "and a screen 110 on which a graphical user interface (GUI) is implemented")

This passage teaches that the computer system which runs the code which generates

the source code objects itself has a GUI and not that the automatically generated source code objects control a computer to display a GUI through which the services in the infrastructure can be invoked.

In other words, this passage does not teach that the code automatically produced by the disclosed invention has a GUI, only that the computer that runs the Goodwin invention may have one.

> (Col. 10, lines 57-59)
> "The repository adaptor tool 312 provides a graphical user interface and framework for collection of adaptors that translate various logical models into unified models"

Again, this passage teaches that a tool or component of the Goodwin invention (namely, the "repository adaptor tool") provides a GUI in one embodiment of said invention. This is not a teaching that the code automatically produced by the disclosed invention has a GUI, only that an embodiment of one component of the computer system on which the process of the Goodwin invention runs may have one.

Collecting service arguments

Claim 1includes the following step regarding creating code which automatically collects service arguments from the formal specification:

> using a computer, automatically write code that will retrieve service arguments for all services from a user or from another object server or from another process, as appropriate;

Goodwin does not teach the automatic generation of code of a software system that collects the arguments of a service (either from a user, another object server or another process) so as to invoke said service. Specifically, at Col. 6, Lines 37-41, Goodwin teaches that the disclosed invention allows developers to generate "objects based on a framework of services" and how said services are authored ("by composing services based on the object templates"), but he does not teach that said generated objects include code that collects the arguments of a service so as to invoke said service.

Matching requested service against server objects implementing it

Claim 1 includes the following step:

> using a computer, automatically write computer code which queries said

specification for all services of all classes that any authorized user may invoke and identifies an object server which will implement each said service;

Goodwin does not teach the automatic generation of code of a software system that matches service requests against the server objects that implement said service.

Checking the validity of state transitions

Claim 1 includes a step:

using a computer, automatically write code that implements an object server for every service, each of which first checks to verify that state transitions are valid and make sense for the current state of objects of which the object service will be altering the state ~~of~~;

Goodwin does not teach the automatic generation of code of a software system that checks the validity of state transitions of objects whose state is changed by the occurrence of services. Col.6, lines 64-67 to col.7, lines 1-7 teach that a component of the disclosed invention is "a data server ... to which users can submit object queries". This does not teach, however that the information retrieved by said queries is used in any way to check the validity of a state transition produced by the occurrence of a service. In contrast, in the method of claim 1 at bar, the set of valid state transitions for objects of a class are defined by means of state machines. No evidence of the use of state machines can be found in Goodwin.

Verifying preconditions

Claim 1 contains a step:

using a computer, automatically write code for every object server that verifies preconditions are satisfied before making state transitions of any objects the states of which are acted upon by the object server;

Goodwin does not teach the automatic generation of code of a software system that verifies preconditions are satisfied for every service that produces a state transition. A precondition is a well-formed formula stating a condition on the state of an object that must be satisfied before the execution of a service which will modify the state of said object, as disclosed in the prosecuted application. What Goodwin teaches in Col. 7, Lines 40-50 is a mapping of objects to data resources, not the automatic generation of code of a software system to verify a sort of condition prior to the execution of services. Moreover, no evidence on the use of preconditions or similar mechanism associated to services can be found disclosed in Goodwin.

Performing valuation calculations

Claim 1 contains the following step:

> using a computer, automatically write code to make all valuation calculations required by said formal language specification of each object server;

Goodwin does not teach the automatic generation of code of a software system that performs the calculations dictated by valuation formulas. As disclosed in the specification at bar, a valuation is a well formed formula that defines the value a variable attribute of an object of a class will have upon occurrence of an event. What Goodwin teaches is one step of the method to "provide the clients periodically updated query-based views of distributed heterogeneous databases" (see col. 16, lines 62-64) or, in other words, provide "a uniform, object-oriented access to legacy data sources by acting as an object-oriented dynamic front into existing databases" (see col.16, lines 59-61).

Therefore, the system disclosed by Goodwin teaches here a mechanism to provide an object-oriented view of a database.

In contrast, the claimed invention of claim 1 teaches the automatic generation of code of a software system to perform the calculation of valuations. That means, the code produced automatically by the process of claim 1, when in execution, controls a computer to determine the values of variable attributes of objects of classes upon the occurrence of events by making the calculations specified in the valuation formulas specified in the formal model.

No evidence on how the values of attributes is changed by the generated code can be found in Goodwin.

Checking integrity constraints

Claim 1 contains the following step regarding checking integrity constraints:

> using a computer, automatically write code to verify that integrity constraints specified in said formal language specification on the values of attributes of objects have been satisfied after execution of a service and take action if said integrity constraints are not satisfied; and

Goldberg does not teach the automatic generation of code of a software system that checks integrity constraints.

The specification of the patent application at bar discloses integrity constraints as being well formed formulas that define a condition on the values of attributes of objects of a class, said condition having to hold for every object of a class to guarantee the

integrity (hence, the name "integrity constraints") of the state of objects.

What Goldberg teaches in Col. 3, Lines 46-50 is a method to verify "that an SQL query created by a user is valid" (not a method to guarantee the integrity of the state of objects) and does so by "submitting the query to the database engine of the underlying database" which means that what is actually checked is the validity of an SQL sentence in terms of the elements (tables, fields) present in the database to which that sentence is addressed. The concepts of integrity constraint, constraint or data integrity are not taught in Goldberg and therefore it does not teach nor suggest this integrity constraint checking element of claim 1 either alone or combined with Goodwin since Goodwin is also silent on this issue.

Checking trigger relationships

> Claim 1 includes the following step regarding trigger relationships:
> using a computer, automatically write code for every object server to test trigger relationships specified in said formal language specification after execution of a service and carry out appropriate action if a trigger event has occurred.

Tse does not teach the automatic generation of code of a software system that check trigger relationships after execution of a service and carry out appropriate action if a trigger event has occurred.

The specification of the application at bar discloses trigger relationships as a pair <condition, service> defined for the objects of a class so that the service is automatically executed whenever the condition becomes true for any object of said class.

What Tse teaches in Col.1, Lines 8-10 is a set of "methods and apparatuses for automatically testing software products", whereas the application at bar teaches automatically writing code of a software system that will execute services whenever a condition is satisfied on the state of an object. The application at bar does not teach nor claim any method or apparatus to test software products.

The concept of trigger relationships is not taught in Goldberg either, and therefore the combination of Goodwin, Goldberg and Tse are still lacking knowledge needed to make the invention, i.e. a method step to automatically write code "for every object server to test trigger relationships specified in said formal language specification after execution of a service and carry out appropriate action if a trigger event has occurred"

SUMMARY OF THE NON OBVIOUSNESS ARGUMENT REGARDING CLAIM 1

Claim 1 is directed to a method to reads a formal language specification and then

automatically write a complete software system that implements the requirements of the formal language specification and does not need any additional software to be added to it with the automatically written code performing the following steps when executed:

- User authentication and validation
- Dynamically determine user privileges as to what information the user is allowed to query and which services the user is allowed to execute
- Display a GUI to invoke services
- Collect service arguments
- Match requested service against service objects implementing it
- Check the validity of state transitions
- Verify preconditions
- Perform valuation calculations
- Check integrity constraints
- Check trigger relationships

The combination of Goodwin, Goldberg and Tse fails to make a prima facie case because even if there is no technological incompatibility in combining the teachings of these three prior art patents, the combination fails to teach all the knowledge needed to make the invention of claim 1. Specifically:

Goodwin does not teach the automatic generation of software systems, nor he discloses the generation of code performing the aforementioned steps, as will be illustrated below. Goodwin only teach automatic generation of software source code objects which are not a complete system and which need to be integrated into a framework of services authored by the user to form a complete system.

Goldberg does not teach how to check integrity constraints, therefore it would not be obvious to one of ordinary skill in the art at the time the invention was made to combine Goldberg and Goodwin to automatically produce code to check integrity constraints. This is because the combination would not creates a full and complete software application, and the source code objects which would be automatically produced would not check integrity constraints.

Tse does not teach how to check trigger relationships, therefore it would not be obvious to one of ordinary skill in the art at the time the invention was made to combine Tse and Goodwin to automatically produce code to check trigger relationships.

In other words, there is no suggestion to make the claimed combination because

there is no perceived liklihood of success in solving the problem the invention solved. This is because one of ordinary skill in the art familiar with the teachings of Goodwin, Goldberg and Tse would recognize that even if the combination were to be made, the result would not be able to function as a complete software system which was able to check integrity constraints and check trigger relationships. Where there is no suggestion to make a combination, there is no obviousness.

The Indefiniteness Rejections

In response to the rejection of claim 1, line 15 on page 94, the following changes have been made:

> using a computer, automatically write code that will retrieve service arguments for all services from one or more of a user, an ~~or from another~~ object server, and ~~or from~~ another process, as appropriate;

These changes make is clear that the service arguments can be retrieved from any one or more of the user, an object server and/or another process.

In response to the rejection of claim 1, line 17-18, the following amendments have been made:

> using a computer, automatically write code that controls a computer to display ~~displays menus options, icons or creates any other~~ means by which and entity ~~a user or another process~~ can invoke a service, and which receives input to invoke a particular service and responds by sending a message to the appropriate object server to invoke the service, said message including the necessary arguments for the service to execute;

These amendments make it clear that the invention automatically writes code which controls a computer to display things which can be invoked by an entity to invoke a service.

Claim 2 was rejected for indefiniteness on the same grounds of indefinite use of "or". As a result, claim 2 was amended to remove the offending or limitation and to specify that the service arguments are retrieved from one or more of a list of three sources.

A number of other changes were voluntarily made to claim 2 to improve its form. For example, the computer was specified as being programmed with an operating system as well as one or more other programs to control it to perform the software writing process.

The preamble was changed to specify that it a formal language specification which is being converted into a computer program. Other changes were made to conform the main body of the claim to the change to "formal specification" in the

preamble.

Similar changes were made to claim 3 to overcome the indefiniteness rejection and

Respectfully submitted,

Dated: October 7, 2004

Ronald Craig Fish
Reg. No. 28,843
Tel 408 778 3624
FAX 408 776 0426

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail, postage prepaid, in an envelope addressed to: Commissioner for Patents , P.O. Box 1450, Alexandria, Va. 22313-1450.
on 10/7/2004
(Date of Deposit)

Ronald Craig Fish, President
Ronald Craig Fish, a Law Corporation
Reg. No. 28,843